Proposal for SMB Direct in Samba by Stefan Metzmacher (2019-07-08)

TOC

Requirements (v4)

```
As the success of implementing SMB-Direct (unlike other typical SMB features)
is bound to the available Hardware and also the Linux Kernel (including the
device specific driver), we need to clarify the project boundaries in order
to build a common expectation of project's outcome.

As each vendor will most likely have a specific hardware setup
that should work in the end, we propose that each vendor provides
a hardware setup that can be used during the development.

The requirements for each setup are these:

5 x Hardware Computers with:
- remote KVM-over-IP access in order to control the
  BIOS/boot sequence and allow forced hardware resets.
- 2 should be servers, similar to the product the
  vendor will later ship, they might have fast
  disks (ssd/nvme)
- 2 can be typical client machines
- 1 will be used to compile and monitor
- 1 server and 1 client and the monitor host
  will run with Linux
  (Ubuntu-18.04 (or newer) amd64 at least in the beginning,
   maybe with dual boot to the vendor's favorit distribution)
- 1 server and 1 client will run with Windows
```

  (I'd use Windows Server 2019 Datacenter on both,
   maybe dual boot to 2016 and/or 2012R2)
- All of them are reachable via ssh or rdp on
  a normal (at least 1 Gbit) network interface from
  the outside.
- The Windows server and client will be taken
  as a reference in order to get a feeling for
  what performance could look like.
- For each R-NIC flavor we need at one card in each
  computer. All 5 cards need to be the exact same model
  with the same firmware version. If more than flavor
  is desired we can have more than one card in each computer.
- Most R-NICs have more than one port, we should use at least
  2 ports. Port 1 of all cards should be attached to the
  same switch, that same applies to port 2 and others.
  If a switch is able to handle 10 ports it should be
  ok to connect all 10 (2x5) ports to the same switch.
- If the interaction with ctdb should be implemented/tested,
  we may need more hardware.

All switches need to offer a monitoring/mirroring port where the
monitoring host can capture traffic between the other hosts,
this is a very important requirement.

We'll start with DiskSpd.exe based tests from the Windows Client,
see how Chelsio did their tests,
https://www.chelsio.com/wp-content/uploads/resources/T5-100G-SMB-Windows.pdf

If the vendor has a special workload in mind the software
needs to be installed on a client (most likely the Windows client)
and run against the Windows server as reference.

While we take Windows as a reference it's very likely its
performance won't be reached (at least initially, but maybe forever).
But the goal is that the SMB-Direct code path is at least not
slower or more cpu consuimg as the TCP code path for the typical
DiskSpd.exe workloads.

The current linux kernel module prototype compiles against
4.10 and higer versions (currently 5.1). Most of the testing
was done with the 4.15 kernel from ubuntu 18.04.

The new io_uring feature of 5.1 (and higher kernels) provides
async io with reduced context switches, we may rely on this
or at least use some concepts. See https://lwn.net/Articles/778411/

As it's the goal to bring the kernel driver upstream (in the long run),
we may only support recent kernel versions (at the time we finalize the project).
Backports to older kernel versions are not part of the project.

As it's not trivial to get kernel changes upstream,
we may have to maintain the out of tree module for a while.
E.g. It took about 9 years for the soft iwarp (siw) driver to
get like to be merged into the 5.3 kernel.

For the Samba part we'll develop against the current master branch.
The Samba-Team has hight standards for getting features pushed
to the upstream master tree (which is the base for future releases).
Typically new features need regression tests. Developing/finishing
such tests and make them part of the pre-push autobuild/gitlab-ci
requires a lot of work (maybe more then developing the features itself!).
This means it's likely to maintain the patches out of the master tree
for a while. Integration of the tests into the ci is not part of
the project.

The maintainance of the out of tree branches is not part of the project. (v2)

Tasks required add SMB 3 SMB-Direct support
(This plan assumes running on a Linux host,
see smb-direct-development-requirements.txt for more detailed contrains!)
This is just the first part that is really required in order
to get something that can be used, e.g. it speaks the
required protocols and it build on top of recent Samba
and Linux-Kernel version (at the time the project finishes).

The feature developement will be designed with upstream
inclusion into Samba and the Linux-Kernel in mind.
But the (most likely very high) required effort for
upstream inclusion is estimated as a separate plan,
which might form a follow-up project. See
"Tasks required bring SMB 3 SMB-Direct to Upstream"
(smb-direct-development-plan-upstream.txt)

1. Multi-Channel

 In order to provide the SMB-Direct feature to Windows clients,
 it's required to finish the Multi-Channel feature in Samba.

 While a lot of things are already in current Samba releases,
 there are still things to do in order to get this into a useable
 state some, which could be considered production ready.

 The detailed estimates can be found in multi-channel-plan.txt,
 for SMB-Direct we require the following tasks:

   1. Replay/Retry Detection
   1.1 locks:
       ==> 1-2w
   3. Multi-Channel
   3.1 Server-Client retry
       ==> 2-3w
   4. Multi-Channel for ctdb cluster nodes
   4.1 interface discovery in a cluster
       ==> 0.5w
   4.2 Session teardown on ip movement
       ==> 0.5w

 The remaining tasks e.g. the Socket-Wrapper changes
 are only required in order to add regression tests into
 the upstream Samba releases and activate the
 "server multi channel support" by default.

==> 4-6w (~1-1.5M)

2. Linux Kernel Driver

There're userspace libraries (libibverbs and librdmacm), which offer
support for RDMA communication, while bypassing the kernel. But
these libraries don't support fd-passing, which is used by Samba
in order to support Multi-Channel.

There was a userspace SMB-Direct proxy daemon, but it's performance
is very bad.

The current design consists of a Kernel driver that provides
a socket interface with some additional features for RDMA data
transfers.

The SMB3 client in the Linux Kernel has experimental support
for SMB-Direct (since 4.15). The new SMB-Direct driver
can also be used there, which might help to get it merged upstream.

There's currently work in progress for
ibv_export_to_fd/ibv_context_to_fd/ibv_import_pd, see
https://marc.info/?l=linux-rdma&m=156155921212805&w=2,
but having just one optimized smbdirect implementation
on Linux (for userspace and kernel consumers) still seems
to be the better solution. Samba needs to serve files
from the kernel space and a kernel driver is likely to
allows less memory copies of the file data.

For now we use /dev/smbdirect and custom ioctl() calls
instead of adding new syscalls.

2.1 Implementation in the Kernel (basics)

There's a prototype that is mostly functional, but lacks

a lot of error checks. It is full of memory leaks
and some features are not yet implemented completely
or only in a sync fashion.

Complete the existing prototype with correct error checking.

Make sure the Windows SMB-Direct testsuite passes against
Samba.

Add useful trace points in order to debug the driver and
find possible performance bottlenecks.

Use a ring buffer interface similar to the io_uring
(See https://lwn.net/Articles/778411/).
Maybe add something like an async ioctl to the uring
interface or true async sendmsg(MSG_OOB).

Experiment/research regarding usage of (transparent) huge pages
and ways to use just one memory registration for an 8MByte
buffer.

We need to batch ib_post_send() requests and only
ask for completions on the last send that belongs
to a higher level operation.

ib_dma_map_sg() merges elements and we may be able to
reduce the size of the smbdirect_buffer_descriptors_v1
array. Each element can hold max_frmr_depth sgl entries
Using just one allows the SEND_WITH_INV to work
effectivelly. TODO: We need to check iov_iter_npages(INT_MAX)
and iov_iter_get_pages() can get all pages of the iter.

Add some restrictions with io_account_mem() and unprivileged
port numbers.

==> 3-4w

2.2 Userspace API for smbdirect:

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

The simplified API used in the current prototype looks like this:

```
 int smbdirect_socket(int family, int type, int protocol);
 int smbdirect_connection_get_parameters(int sockfd,
                                         struct smbdirect_connection_parameters *params);
 ssize_t smbdirect_rdma_v1_register(int sockfd,
                                    struct smbdirect_buffer_descriptors_v1 *local,
                                    int iovcnt, const struct iovec *iov);
 ssize_t smbdirect_rdma_v1_deregister(int sockfd,
                                      const struct smbdirect_buffer_descriptors_v1 *local);
 ssize_t smbdirect_rdma_v1_writev(int sockfd,
                                  const struct smbdirect_buffer_descriptors_v1 *remote,
                                  int iovcnt, const struct iovec *iov);
 ssize_t smbdirect_rdma_v1_readv(int sockfd,
                                 const struct smbdirect_buffer_descriptors_v1 *remote,
                                 int iovcnt, const struct iovec *iov);
 void smbdirect_buffer_invalidate_cmsg_prepare(struct msghdr *msg,
                                  uint8_t *buf,
                                  size_t buflen,
                                  const struct smbdirect_buffer_descriptor_v1 *desc);
```

The created socket provides a stream with 4 byte (big endian) length delimiters,
in order to provide a compatible with SMB over TCP.

The smbdirect_buffer_descriptors_v1 based APIs should be converted to mmap'ed
ring buffer interface, similar to io_uring. At least the writev/readv calls
need to be async.

An optimized version with sendfile/recvfile performance like this:

```
 ssize_t smbdirect_rdma_v1_write_from_file(int sockfd,
                            const struct smbdirect_buffer_descriptors_v1 *remote,
                            int source_fd, size_t source_length, off_t source_offset);
 ssize_t smbdirect_rdma_v1_read_to_file(int sockfd,
                            const struct smbdirect_buffer_descriptors_v1 *remote,
                            int source_fd, size_t source_length, off_t source_offset);
```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

These would be implemented using in kernel or userspace bounce buffers in the first version,
later we may be able to use the page cache directly or even rdma directly
from the block device.

We also need ways to specify parameters like timeouts, max credits, max io size
and more before connect/listen. So these can be specified per socket from
userspace, as it makes it easier to find the optimized values.

The following is required for smbd to announce RDMA interfaces.

```
 int smbdirect_netif_rdma_capable(const char *ifname);
```

==> 1w

2.3 Kernelspace API for smbdirect

The simplified API used in the current prototype looks like this:

```
 int smbdirect_sock_create_kern(struct net *net,
                                int family, int type, int protocol,
                                struct socket **res);
 int smbdirect_kern_connection_get_parameters(struct socket *sock,
                                              struct smbdirect_connection_parameters *params);
 ssize_t smbdirect_kern_rdma_v1_register_pages(struct socket *sock,
                                               struct smbdirect_buffer_descriptors_v1 *local,
                                               struct page *pages[], int num_pages,
                                               int pagesz, int fp_ofs, int lp_len);
 ssize_t smbdirect_kern_rdma_v1_deregister(struct socket *sock,
                                               struct smbdirect_buffer_descriptors_v1 *local);
```

Most likely we can keep this api, if really required
smbdirect_kern_rdma_v1_deregister can be made async.

Convert fs/cifs/ to use this api as alternative to
fs/cifs/smbdirect.c and compare the performance results.

This is not strictly required, but it will help in order
to find bugs and compare the functionality and performance
with the existing fs/cifs/smbdirect.c implementation of
SMB-Direct.

==> 1-2w

2.4 Write a standalone testsuite

This should not use SMB3, but a small custom
echo-like protocol that is able to test the
api and protocol.

The protocol would have opcodes like this:

```
 REQ_HEADER {
     __le32 message_id;
 };
 REP_HEADER {
     __le32 message_id;
     __le32 status;
 };
 DESCRIPTOR {
     __le64 iova;
     __le32 rkey;
     __le32 length;
 };
 ECHO-REQ(REQ_HEADER hdr,
          __le16 echo_factor,
          __le32 data_len,
          __le32 max_response,
          u8 data[date_len])
 ECHO-REP(REP_HEADER hdr,
          __le32 data_len,
          u8 data[data_len])
 /*
  * SETUP-LBUF creates a buffer on the
```

```
  * server where the content will be
  * for (ofs=0; ofs < size; ofs += 8) {
  *      PUSH_UINT64_LE(buf, ofs, nonce ^ ofs);
  * }
  * This simulates a file, the server maintains
  * an array of LBUF (files), lbuf_idx is the index into
  * that array.
  */
 SETUP-LBUF-REQ(REQ_HEADER hdr,
                u32 lbuf_idx,
                u32 size,
                u64 nonce);
 SETUP-LBUF-REP(REP_HEADER hdr)
 READ-LBUF-TO-DESCS-REQ(REQ_HEADER hdr,
                        u32 lbuf_idx,
                        u32 flags, /* SEND_WITH_INV */
                        u32 offset,
                        u32 length,
                        u32 num_descs,
                        DESCRIPOR descs[])
 READ-LBUF-TO-DESCS-REP(REP_HEADER hdr,
                        u32 nread)
 WRITE-DESCS-TO-LBUF-REQ(REQ_HEADER hdr,
                         u32 lbuf_idx,
                         u32 flags, /* SEND_WITH_INV */
                         u32 offset,
                         u32 length,
                         u32 num_descs,
                         DESCRIPOR descs[])
 WRITE-DESCS-TO-LBUF-REP(REP_HEADER hdr,
                         u32 nwritten)

It can be used as regression and performance test
for the kernel parts without having to deal with
Samba or the fs/cifs/ code.

==> 1w
```

==> 6-8w (~1.5-2M)

3. Add support for SMB-Direct to Samba

3.1 Add support to the generic client library

There's already a prototype to let 'smbclient'
use SMB-Direct with RDMA. But it's not used
automatically as multi-channel support with
interface detection is missing.

For a start we need to be able to write tests
and allow smbclient to specific the transport
protocol e.g. NetBios, TCP, SMB-Direct (or later QUIC).

This is required in order to test without
relying just on highlevel tests with Windows.

We also need to write some smbtorture tests.

==> 2-3w

3.2 Add support to smbd READ/WRITE

There's already a prototype that allow connections
via SMB-Direct including RDMA read and writes, but
they use a sync interface in order to do the RDMA
read/write operation these need to make async.

We should maybe able to implement some optimization
like sendfile() in order to avoid data copies.

A custom ioctl on the SMB-Direct socket or
some interaction with io_uring and linked
operations are possible solutions.

Can we do RDMA read/writes from/to the page cache
of files?

==> 1-2w

3.3 Plug SMB-Direct into the interface detection

We need a way to specify if smbd should
listen on SMB-Direct sockets. If so we should
automatically mark interfaces exported by
the Multi-Channel interface detection as RDMA capable.

==> 1w

==> 4-6w (~1-1.5M)

4.1 Analyze Hardware related problems

In the past we hit very strange problems where
the exact same software provided completely different
performance results on the same hardware one day later.

==> 1-2w

4.2 Fix bugs/interop problems/missing features in generic kernel code

There're problems with the RDMA READ responses from a Linux client
against a Windows Server 2016 (with Chelsio T404-BT).
Windows only accepts responses up to 16204 bytes fragments.
At least with SoftIWARP on the client. Another problem similar
happens with Chelsio T404-BT on the client.

We may hit other problems during development.

 ==> 1-2w
==> 2-4w (~0.5-1M)
Total: ==> 16-24w (~4-6M)

Tasks required add SMB 3 MultiChannel support (v8)

(This plan assumes running on a Linux host)

A lot of code is already in existing Samba versions,
but some critical parts are still work in progress, see
https://git.samba.org/?p=metze/samba/wip.git;a=shortlog;h=refs/heads/master3-multi-channel

The Replay/Retry detection is a requiement in order
to make it safe to offer multi-channel support to clients.

We need to socket_wrapper enhancements in order to
add automated regression tests in Samba's master branch.

Note that a lot of the estimated time will be consumed
by manual testing/research and writing automated tests.

1. Replay/Retry Detection

1.1 locks:
~~~~~~~~~~
  - "LockSequence" number (in SMB2 Lock request) uniquely identifies (un)lock
    request among all (un)lock requests to the same file
  - applies to SMB version >= 2.1
    resilient handles, multi channel, persistent, ...
  - array of 64 lock requests per open on client and server
    - client can only have 64 outstanding lock/unlock requests per open)
    - bucket index = index (0..63) into array of outstanding lock requests
      (bucket)
    - bucket number = bucket index + 1
    - client sends lock sequence = (bucket number << 4) + mod 16 incrementing sequence number
    - server stores the sequence number in LockSequenceArray
~~~~~~~~~~

```
      by reversed calculation after successful lock processing:
      - index = (locksequence >> 4) - 1
      - sequence = least 4 bits of lock sequence
    - if the server recieves a lock request with sequence already
      existing in the array, it simply replies with success
  - server implementation: simple
  - TEST:
    - write tests in smbtorture and on windows to verify behaviour
  - TODO: ask dochelp:
    - about initialization with 0xFF instead of 0x00
    - about scope (resilient, leasing, ... + ?)
  ==> 1-2w

==> 1-2w (~0.25-0.5M)

2. Socket-Wrapper
-----------------

Socket wrapper maintains a socket_info structure for each
the represents a tcp/udp socket. As it's possible to have more
than one file descriptor for a socket, there can be multiple
socket_info_fd structures pointing to a single socket_info structure.
Currently socket wrapper maintains a global linked list,
without any thread safety.

Socket wrapper doesn't support fd-passing of tcp/udp sockets,
the fd is passed, but it appears as unix domain socket in the
other process.

2.1 maintain a "db file" for the socket_info structures
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

  - We need to maintain a small file using mmap and protected
    by pthread robust mutexes. E.g. one file per local ip address.
  - The path specified in SOCKET_WRAPPER_FD_PASSING_DB will
    be used as the file name, if this is not specified we'll
    use malloc'ed and fd-passing is not enabled.
  - The file contains a header (with magic, unique id, size and free-list pointer)
```

```
    followed by an array of socket_info structures.
  - The socket_info_fd structures will only maintain the index
    into the mmap'ed array.
  - fd-passing is limited to fixed number (~ 127), this should be
    more than enough for typical caller (Samba would just use 1).
  - In order to do fd-passing of tcp/udp sockets, we'll create
    a pipe (or similar) where we write an array of with indexes
    into the mmap'ed array into the write end of the pipe.
    We would also pass the device/inode and a unique identifier
    for the file.
    The read end of the pipe is then passed as the last fd to
    the destination process. The destination process can rebuild
    the socket_info_fd structures by reading the array indexes.
    out of the read end of the pipe.
  - A tricky part will be the reference counting in the database
    entries. The sender needs to write the data into the pipe
    and increment the refcounts in the db file before calling
    sendmsg(). The sender may hold a mutex for each socket
    during sendmsg().
  - In order to allow multiple threads (or processes) to share a single
    socket we need to add mutex protection in quite a few places.
    In the most common cases there won't be any contention on the mutexes,
    but it will garantee correctness for the corner cases which happens
    for fd-passing.
  - See https://gitlab.com/anoopcs/socket_wrapper.git
        gitlab-anoopcs-socket_wrapper/fd-passing-final
  - See https://gitlab.com/metze/socket_wrapper.git
        gitlab-metze-socket_wrapper/fd-passing
  - TESTS
  ==> 2-3w

==> 2-3w (~0.5-0.75M)

3. Multi-Channel
----------------

scope:
~~~~~~
```

- SMB-only
- No ctdb cluster (see 4.)

3.1 Server-Client retry
~~~~~~~~~~~~~~~~~~~~~~

  - enable keepalives
  - Functions like getsockopt(TCP_INFO), ioctl(SIOCINQ), ioctl(fd, SIOCOUTQ) or
    ioctl(SIOCOUTQNSD) are hopefully able to detect if some bytes are already
    (tcp) acked by the client.
  - retry to send Oplock/Lease Breaks on a different channel
    (if there is more than one channel)
  - See https://bugzilla.samba.org/show_bug.cgi?id=11897
  - See https://github.com/spuiuk/samba.git
    github-spuiuk/sp-multichannel.20190618
  - TESTS
  - TODO: ask dochelp:
    - is it ok for Windows client to get multiple oplock/lease breaks
  ==> 2-3w

==> 2-3w (~0.5-0.75M)

4. Multi-Channel for ctdb cluster nodes
----

scope:
~~~~~~
- The interaction with ctdb cluster
- multiple channels on a single node, not on multiple nodes simultaneouly
  (this matches the behaviour of Windows SOFS clusters)
- in a ctdb cluster, we should make sure that we only
  return ip addresses local to the node.
  possibly ctdb needs change / be configured to
  not handle public addresses.
- one client will only ever connect to one cluster node.

4.1 interface discovery in a cluster
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

```
 - The implementation of FSCTL_QUERY_NETWORK_INTERFACE_INFO
   should use CTDB_CONTROL_GET_PUBLIC_IPS and blacklist
   all public addresses from the returned response.
 ==> 0.5w

4.2 Session teardown on ip movement
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

 - If smbd gets a CTDB_SRVID_RELEASE_IP message
   from ctdbd for an ip that's attached to a
   ctdb public address, it should shutdown all
   connected channels instead of just the one
   attached to the specific ip address.
   This means the client needs to reconnect completely
   (most likely to another node) and will ask for
   available MultiChannel interfaces again.
 ==> 0.5w

==> 1w (~0.25M)

Total: ==> 6-9w (~1.5-2.25M)
```

Tasks required bring SMB 3 SMB-Direct to Upstream (v1)

```
(This plan assumes running on a Linux host,
see smb-direct-development-requirements.txt for more
detailed contrains!)

This is just the 2nd part that is based on
"Tasks required add SMB 3 SMB-Direct support"
(smb-direct-development-plan-usable.txt)

5. Upstream Samba Changes
-------------------------

5.1 Automated Multi-Channel Testing
```

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

 We need to have regression tests in our autobuild/gitlab-ci
 in order to remove the "experimental" flags from
 the "server multi channel support" option and enable it
 by default.

 The remainig blocker is fd-passing support in Socket-Wrapper.

 The detailed estimates can be found in multi-channel-plan.txt,
 for SMB-Direct we require the following tasks:

   2. Socket-Wrapper
   2.1 maintain a "db file" for the socket_info structures
       ==> 2-3w
   ==> 2-3w (~0.5-0.75M)

 Note that Multi-Channel is not a Linux-specific feature, which
 means we can't rely on Linux containers/namespaces.

 ==> 2-3w

5.2 Automated SMB-Direct Testing
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

 Samba requires features to be tested by every push
 to the upstream repositories.

 There is support for using network namespaces
 (on Linux) instead of socket wrapper for testing.

 Maybe it's possible to integrate something like this
 into the gitlab-ci. We could download the kernel
 driver sources and build/load the kernel module
 before starting the SMB-Direct specific tests
 inside a KVM machine (it might not be possible
 to do that from within a docker container).

Or we just have a stable version of the
kernel driver installed on a private runner.

==> 2-5w

==> 4-8w (~0.5-0.75M)

6. Linux Kernel Driver

6.1 Optimizations in the kernel driver

Possibly required performance tunning can (as always) take
an infinite amount of time. But we already got a lot of useful
hints from Microsoft what aspects are relavant for good performance.

We may need to add some cache line alignment.

We may need to add numa related changes and pin
kernel threads to specific cpu's in the same numa domain.
Maybe blk_mq_rdma_map_queues() and ib_get_vector_affinity()
provide some hints?

TODO: DIM (dynamic interrupt coalescing)
https://marc.info/?l=linux-rdma&m=156161312826038&w=2

==> 2-4w (maybe more)

6.2 Upstream integration

Make use of the correct kernel APIs.

The initial design is based on /dev/smbdirect
and ioctl() calls to create and operate on a socket.

In order to bring this code upstream it's very likely

that we have to adopt to modern kernel APIs.

It's very likely to take lot of time to
discuss with the kernel community about
the correct ways to integrate the driver into
the upstream kernel.

We may also have to rewrite parts of the driver
in order to make it acceptable.

It might be easier to propose only the in kernel
api first and let replace fs/cifs/smbdirect.* to be
used in fs/cifs. Small selfcontained chunks are
much easier to review and have a much higher chance
to get accepted.

Here is just a list of things which are likely to
be considered/researched/discussed:
- Should we introduce a PF_SMBDIRECT to
  replace the ioctl() on /dev/smbdirect
  in order to allow the socket() syscall
  to create the socket?
- Should we use getsockopt/setsockopt to replace all/some
  custom ioctl() calls?
- Should we try to integrate our custom async handling
  with io_uring?
- Do we need special handling in order
  to interact with network namespaces
  (See register_pernet_subsys())
  (See "rdma sys set netns shared"?)
- Do we need to interact with netlink sockets?
- Should we use common structures for buffer memory handling
  See sk_buff/sock_kmalloc/sk_receive_queue/sk_wmem_alloc/sk_memcg/
  sk_stream_wait_memory/sk_enter_memory_pressure/sk_prot_mem_limits/
- Can we use sock_def_write_space instead of our own?
- Do we need to implement any/more of common getsockopt/setsockopt
  opcodes?
- Should be make use of sk_stream_error(), see sk_err/sk_err_soft

```
   and SIGPIPE
 - Do want to interact nicely with common tools like
   "netstat" or "ss" and replace the custom diagnostics from /proc/smbdirect?
   See also sock_diag_register/sock_diag_handler.
 - Do we need to implement some of the SK_FLAGS_TIMESTAMP logic?
 - We may need to interact with various security/filter layers
   in the kernel. Research on security_socket_post_create,
   BPF_CGROUP_PRE_CONNECT_ENABLED, netfilter, BFP filters
   and related things.

 While doing the integration work we may find bugs or limitations
 in existing kernel infrastructure, we may need to do some generic
 cleanups in all sorts of kernel subsystems.

 As smbdirect should work with every RDMA hardware or software
 driver in the kernel, we may need to analyse problems related
 hardware or the related driver. We may hit unsolveable problems,
 but in that case we should at least try to fail gracefully
 (at best before establishing a connection). E.g. we rely
 on FRWR (Fast Registration Work Requests) support and check
 for IB_DEVICE_MEM_MGT_EXTENSIONS.

 It's not really possible to estimate the amount of time
 to get things into the upstream kernel.

 E.g. it took about 9 years (real time) from the first proposal
 for the Soft iWarp driver (siw) to be scheduled upstream inclusion
 in the 5.3 kernel. A lot of patience seems to be required here.

 ==> 2-8w (maybe much more)

==> 4-12w (~1-3M) (maybe much more)

7. Future Protocol Features
~~~~~~~~~~~~~~~~~~~~~~~~~~~

7.1 Add signing/encryption support for RDMA read/write
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Microsoft announced that they going to add a new
CHANNEL type (wrapper) and sign/encrypt at the
SMB3 layer before doing the RDMA read/write.

We may want to support that too.

The initial support for this doesn't look to
hard to implement, it can most likely be done
only within Samba.

A more advaced solution would include encryption
to happen within the kernel.

But the specification and reference implementation
(Windows) isn't available yet (planed for Q3 2019),
so it's not possible to estimate the required efforts.

==> n.a.

7.2 Add PUSH-Mode support for RDMA memory mapping

Microsoft discussed PUSH-Mode support for
remote memory mapping of DAX based files.

We may want to support that too.

There is not yet enough information to estimate.

==> n.a.

==> n.a.

Total: ==> 8-20w (~2-5M) (maybe much more)